

# Assignment 1: Complexity and Sorting

## Description

In this assignment, you'll work in groups to create a sorting algorithm for geometric shapes. You'll also implement and perform experimental analysis (benchmarking) on six sorting algorithms.

**Warning:** This project is significantly larger and more complex than the assignments in your previous semesters, do NOT wait until the last week of the deadline to start!

## Equipment and Materials

For this assignment, you will need:

- Eclipse and Java 8

## Instructions

This assignment consists of three parts, to be completed **outside** of class time. See the course outline and Brightspace for due dates. Complete this assignment with your assigned group.

### Part A: Create a Sorting Application (90 marks)

Using the specifications below, create an application to sort any object according to a specific property. You'll then implement and perform experimental analysis (benchmarking) on six sorting algorithms:

- Bubble
- Insertion
- Selection
- Merge
- Quick
- A sorting algorithm of your choice

You must research the sorting algorithm of your choice and ensure that it's **significantly different** from the other five. This must be an algorithm that does complete sorting the test files within a reasonable amount of time! (e.g. no bogosort!) Include a detailed description of this sort's algorithm and a complexity analysis outlined in the *Submission Deliverables* section of this document.

**Note:** If you're unsure of the algorithm you're allowed to choose, please discuss it with your instructor!

## Part B: Complete an Application Evaluation as a Group (5 marks)

After completing your sorting application, check your work against the provided marking criteria. Your instructor will refer to your group's self-evaluation when grading the assignment and will provide further feedback and grade adjustments as needed. Your instructor is responsible for awarding the group's final grade.

1. Open the Marking Criteria document (MarkingCriteria\_Assignment1.docx) and save a copy with your group's name.
2. As a group, discuss how well you met each criterion and assign yourselves a mark for each row in the table. You may include a short, point form, explanation for your mark in the Notes column.

**Note:** This is an opportunity for you to identify any missing pieces according to the marking criteria, make sure you do this step diligently to avoid losing any marks that is clearly stated in this document.

3. Save this file for submission to Brightspace along with your completed code.

## Part C: Complete a Peer Assessment (5 marks)

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

## Assignment Specifications

**Important:** Read the specifications very carefully. If you are uncertain about any of the requirements, discuss it with your instructor.

4. Import the assignment1StartingCode project into Eclipse.
5. Create an abstract class in Java that represents a three-dimensional geometric shape. Using the Strategy pattern discussed in class:
  - Implement the **compareTo()** method of the Comparable interface to compare two shapes by their height, and the **compare()** method of the Comparator interface to compare two shapes by their base area and volume.
    - Your implementations of these interfaces **MUST** adhere to the contract! (i.e. return values as specified: negative, positive or zero)
  - The compare type will be provided as input from the command line to your program: **h** for height, **v** for volume, and **a** for base area via the -t option.
6. Write a testing program that will read a text file (entered at the command line via the -f option) of random shapes that adds them to an array (not ArrayList).

---

**Notes:**

- All shapes must be manipulated as elements of the corresponding collection.
  - The value in the first line of the data file contains the number of shapes in that file.
  - Each subsequent line in the data file will contain the values for each shape that you will need to generate the corresponding object.
  - A shape in the file is represented as follows on each line (the values are separated by spaces):
    - One of: Cylinder, Cone, Prism or Pyramid
    - Cylinders and Cones are followed by a double value representing the height and another double value representing radius.  
e.g., Cylinder 9431.453 4450.123 or Cone 674.2435 652.1534
    - Pyramids are followed by a double value representing the height and another double value representing edge length.  
e.g., Pyramid 6247.53 2923.456
    - Prisms are specified by the type of base polygon (SquarePrism, TriangularPrism, PentagonalPrism, OctagonalPrism), a double value representing the height and another double value representing edge length.  
e.g., SquarePrism 8945.234 3745.334
  - See the formulas at the end of this document for information on how to calculate the base area and volume for each type of shape.
7. Your testing application should then invoke the utility methods to re-arrange the shape objects according to the compare type from the **largest to smallest** (descending order).
- Note:** Sorting in ascending order and then reversing the array is NOT acceptable! Your sorting algorithm must use the comparable and comparators correctly to re-arrange the objects.
8. Your testing program should print the time that it took to sort the collection of objects for each of the six sorting algorithms, including a measurement unit (e.g., milliseconds). This should ONLY include the time for the sorting algorithm to run – not including the time to read the input file, generate the objects, or display the objects' data to the screen.
- Note:** Have this benchmarking done outside of the sorting algorithms such that the sorting code can be executed without this explicit output.
9. The program should also display to the console – the first sorted value and last sorted value, and every thousandth value in between, for example:

```

First element is:      shapes.PentagonalPrism      Height: 39999.876
1000-th element:      shapes.PentagonalPrism      Height: 37983.006
2000-th element:      shapes.OctagonalPrism        Height: 36056.608
3000-th element:      shapes.Cylinder              Height: 34067.854
4000-th element:      shapes.PentagonalPrism        Height: 32134.6
5000-th element:      shapes.Cylinder              Height: 30213.166
6000-th element:      shapes.Cylinder              Height: 28290.742
7000-th element:      shapes.SquarePrism           Height: 26243.494
8000-th element:      shapes.Cylinder              Height: 24335.663
9000-th element:      shapes.Cylinder              Height: 22264.57
10000-th element:     shapes.Cone                  Height: 20229.919
11000-th element:     shapes.TriangularPrism        Height: 18208.343
12000-th element:     shapes.PentagonalPrism        Height: 16236.675
13000-th element:     shapes.PentagonalPrism        Height: 14327.071
14000-th element:     shapes.Cone                  Height: 12349.82
15000-th element:     shapes.SquarePrism           Height: 10454.019
16000-th element:     shapes.Cone                  Height: 8406.61
17000-th element:     shapes.PentagonalPrism        Height: 6477.684
18000-th element:     shapes.Pyramid               Height: 4514.668
19000-th element:     shapes.Cylinder              Height: 2557.259
20000-th element:     shapes.Cylinder              Height: 460.886
Last element is:      shapes.Cone                  Height: 2.094
Bubble Sort run time was: 2645 milliseconds

```

```

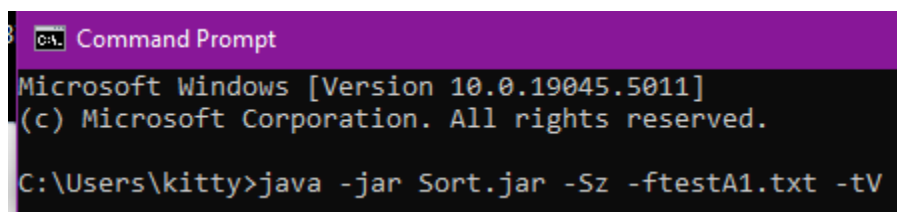
First element is:      shapes.OctagonalPrism        Volume: 3.038614197305051E14
1000-th element:      shapes.OctagonalPrism        Volume: 8.85733916462701E13
2000-th element:      shapes.SquarePrism           Volume: 5.399130444127465E13
3000-th element:      shapes.Cylinder              Volume: 3.785778484832915E13
4000-th element:      shapes.Cylinder              Volume: 2.744012588843761E13
5000-th element:      shapes.Cylinder              Volume: 2.1152523301607426E13
6000-th element:      shapes.SquarePrism           Volume: 1.6362689879264312E13
7000-th element:      shapes.Cone                  Volume: 1.2836684610125104E13
8000-th element:      shapes.TriangularPrism        Volume: 1.0144141146758613E13
9000-th element:      shapes.OctagonalPrism        Volume: 8.031077712122565E12
10000-th element:     shapes.TriangularPrism        Volume: 6.303799517139714E12
11000-th element:     shapes.SquarePrism           Volume: 4.815490542116835E12
12000-th element:     shapes.TriangularPrism        Volume: 3.6181908791370186E12
13000-th element:     shapes.PentagonalPrism        Volume: 2.675147155327021E12
14000-th element:     shapes.TriangularPrism        Volume: 1.9015644218194382E12
15000-th element:     shapes.Cone                  Volume: 1.3083932010669653E12
16000-th element:     shapes.Cone                  Volume: 8.165810615734723E11
17000-th element:     shapes.Cone                  Volume: 4.656736698613374E11
18000-th element:     shapes.Cone                  Volume: 2.1136211810450534E11
19000-th element:     shapes.Cone                  Volume: 6.206127129334734E10
20000-th element:     shapes.Cone                  Volume: 2.5383875178680935E9
Last element is:      shapes.Cone                  Volume: 161275.57456990532
Insertion Sort run time was: 4982 milliseconds

```

```
First element is:      shapes.OctagonalPrism      Area: 7.719478003776468E9
1000-th element:      shapes.Cone                  Area: 4.520723113243874E9
2000-th element:      shapes.Cone                  Area: 3.3988731109298944E9
3000-th element:      shapes.Cylinder              Area: 2.5518003565408955E9
4000-th element:      shapes.Cone                  Area: 1.976826058911541E9
5000-th element:      shapes.PentagonalPrism        Area: 1.5319014706858184E9
6000-th element:      shapes.OctagonalPrism        Area: 1.3064719839257457E9
7000-th element:      shapes.SquarePrism           Area: 1.100462790554176E9
8000-th element:      shapes.Pyramid               Area: 8.93546968769536E8
9000-th element:      shapes.Cylinder              Area: 7.154517130640285E8
10000-th element:     shapes.Pyramid               Area: 5.90685096512196E8
11000-th element:     shapes.Cone                  Area: 4.8450113699886215E8
12000-th element:     shapes.SquarePrism           Area: 3.8615225067304903E8
13000-th element:     shapes.SquarePrism           Area: 2.9930190251833606E8
14000-th element:     shapes.TriangularPrism       Area: 2.1971920774914467E8
15000-th element:     shapes.TriangularPrism       Area: 1.5476208310016066E8
16000-th element:     shapes.PentagonalPrism        Area: 1.0191796770289664E8
17000-th element:     shapes.Cone                  Area: 6.110438128732696E7
18000-th element:     shapes.Pyramid               Area: 2.9222446596841004E7
19000-th element:     shapes.SquarePrism           Area: 8554542.784225
20000-th element:     shapes.OctagonalPrism        Area: 340580.1305243869
Last element is:      shapes.PentagonalPrism        Area: 7.457794679790608
Quick Sort run time was: 31 milliseconds
```

**Note:** The above does not show the output from the test files provided, it is simply to give you an example of what the output would look like – your results will vary depending on the file being parsed. Your program output should be similar but does NOT have to match the exact formatting as long as the relevant information are shown. Your instructor will use a different test data file to evaluate your application.

10. To run the executable (jar) program, the user should be able to enter the following via the command prompt using the following case insensitive format:



```
C:\Users\kitty>java -jar Sort.jar -Sz -ftestA1.txt -tv
```

```
java -jar Sort.jar -f file_name -tv -sb
```

- -f or -F followed by file\_name (the file name and path) with no spaces
- -t or -T followed by v (volume), h (height) or a (base area) with no spaces
- -s or -S followed by b (bubble), s (selection), i (insertion), m (merge), q (quick) or z (your choice of sorting algorithm) with no spaces

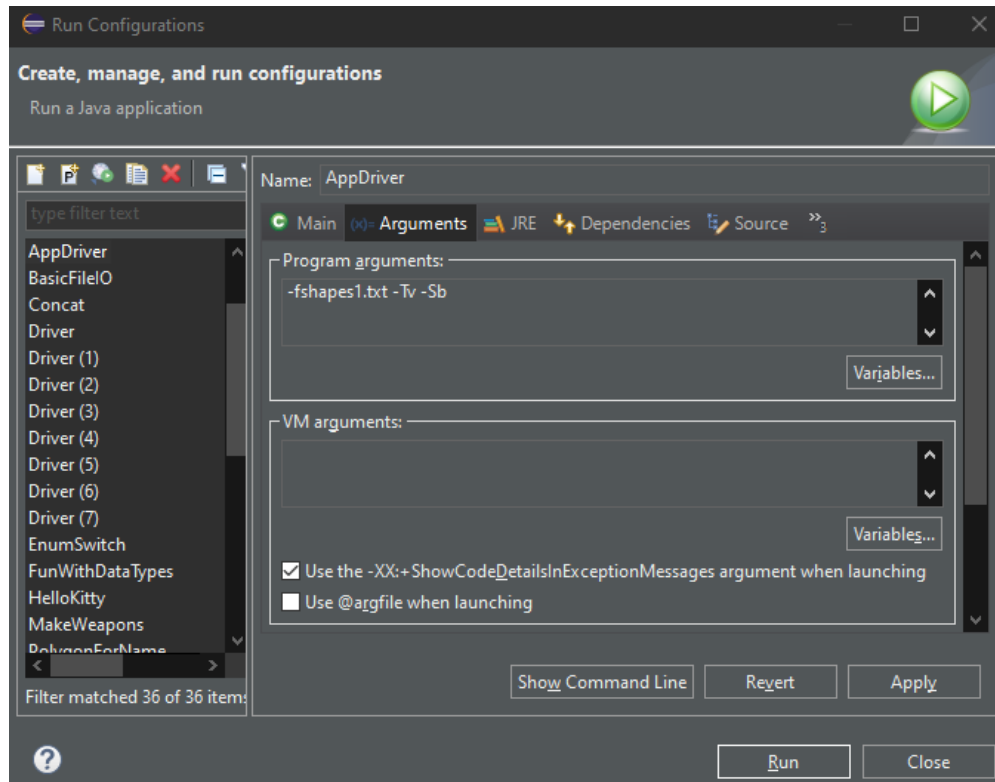
The program must be able to handle these command line arguments no matter the order (order insensitive). The following are some examples of valid inputs:

- java -jar Sort.jar -fshapes1.txt -Tv -Sb

- `java -jar Sort.jar -ta -sQ -f"res\shapes1".txt`
- `java -jar Sort.jar -tH -F"C:\temp\shapes1.txt" -sB`

**Note:** You can assume the user will provide the correct absolute or relative path to the required file as shown in the examples above.

You can test this command input in Eclipse using the “Run Configurations” tool under the “Arguments” tab:



11. If the user enters an incorrect command line argument according to the rules above, the program should display a meaningful message to help the user in correcting the error.
12. You will want to first test your sorting algorithms starting with the smallest data set (shapes1.txt) to check the results.
13. Make sure your code works with the other test data! (i.e., shapes2.txt and shapes3.txt)
14. 😊 Challenge! 😊 Try your code out with the BIG data set ([shapesBig.txt](#))! Does it still run correctly? :D

## Submission Deliverables

**Note:** This is a group assessment, so only one submission is required per group. If there are multiple submissions, your instructor will only evaluate the last submission. See Brightspace for the exact due date and time.

Your group's submission should include a zipped folder named as the assignment number and your group name (e.g., A1Zelda.zip). The zipped folder should contain the following items:

- 1) An **executable** Java Archive file (.jar) for your sorting application called **Sort.jar**.
- 2) A **readMe.txt** file with instructions on how to install and use your sorting program.
- 3) The project should have completed javadoc using the "-private" option when generated. Place the output in the **doc** directory of the project.
- 4) A folder containing the complete export Eclipse project directory. Refer to the exact instructions of how to export your project from Lab 0.

**Note:** Do NOT include any of the test data files in your upload (e.g., shapesBig.txt).

- 5) A **mySort.txt** file (or any document format, e.g., word, pdf etc.) that provides a detailed description of the sorting algorithm of your choice along with its complexity analysis.
  - Your analysis should include the steps of the algorithm in **pseudocode**, with a **counting analysis** (the number of operations performed in each step).
  - **Note:** Make sure to write this in your own words and NOT copy or rephrase from another source.
- 6) The completed Marking Criteria document containing your group's evaluation of your application. All group members should be present when completing this document and it acts as your "sign-off" (approval) of the assignment submission.

**No late assignments will be accepted.**

---

## Formulas

### Cylinder

- $base\ area = \pi r^2$
- $volume = \pi r^2 h$

### Cone

- $base\ area = \pi r^2$
- $volume = \frac{1}{3}\pi r^2 h$

### Pyramid

- $base\ area = s^2$
- $volume = \frac{1}{3}s^2 h$

## Prisms

### Square Base

- $base\ area = s^2$
- $volume = s^2 h$

### Equilateral Triangle Base

- $base\ area\ A = \frac{s^2\sqrt{3}}{4}$
- $volume = Ah$

### Pentagon Base

- $base\ area\ A = \frac{5s^2\tan(54^\circ)}{4}$
- $volume = Ah$

### Octagon Base

- $base\ area\ A = 2(1 + \sqrt{2})s^2$
- $volume = Ah$